

Del 6 – Strömmar

Ämnesområden denna föreläsning:

- Utmatning till `ostream` och `ostream`
- Inmatning från `istream` och `istream`
- Överlagring av inmatning och utmatning
- Iteratorer för in- och utmatning
- Filhantering med strömmar

Slide 1

Exempel på strömmar:

```
#include <iostream>      // innehåller cout
#include <iomanip.h>     // innehåller setw()

cout << "Demo:"        // utmatning av const char *
     << setw(10)       // sätt storlek på utmatningsfält
     << 4711           // utmatning av int
     << endl;         // radmatning

cout << "Name\tPts\n"  // rubriker
     << "Joe\t37\n"    // '\t' är tabulatorstecken
     << "Anna\t41\n";  // '\n' är radmatning

int i, j;
cin >> i >> j;        // hämta int från stdin
```

Slide 2

Utmatning med `cout`, `cerr`

- För utmatning och felutmatning använder man `cout` resp. `cerr`.
- Prefixet *c* står för *character*. Trots detta kan man mata in och ut alla sorters egna och inbyggda typer.
- `cout` och `cerr` är globala objekt med överlagrad `operator<<`

Slide 3

```
std::string str = "C++";
const void *p = "Java";

cout << 7L << " "           // 7           (long)
      << 3.14 << " "        // 3.14        (double)
      << str << " "          // C++         (std::string)
      << str.c_str() << " " // C++         (const char *)
      << p << endl;         // 0x135b0 (adress)
cerr << "error: div by zero" << endl; // till stderr
```

Formatväljare för utmatning

Man kan variera formatet på sin utmatning:

Slide 4

```
double balance = 147.5371; // pengar på kontot
cout.precision(2);         // avrunda till två decimaler
cout << "Balance: "        // rubrik
      << setw(10)           // tio siffrors fält
      << setfill('*')       // fyll tomrum med *
      << balance            // utskrift ger resultat:
      << endl;             // Balance: ****147.54
```

Utmatning till/inmatning från minne

Om man vill skriva till/från minnet istället för fil kan man använda `stringstream` som skriver till/från en STL-sträng.

Slide 5

```
#include <stringstream>

ostreamstream message;          // mata till minne
int i = 17;
double d = 33.3;
message << "Error: " << i      // fyll med data
        << " is less than " << d;
cerr << message.str() << endl; // skriv ut

istringstream source("To be or not to be");
string str;
source >> str;                  // ett ord i taget
```

Inmatning med `cin`

För inmatning från `stdin` använder man `cin`. Typen på argumentet till `operator>>()` bestämmer hur indata tolkas.

Slide 6

```
// antag inmatning för ett bankkonto:
// kalle qwerty 147.30
string user, passwd;
double balance;

cin >> user >> passwd >> balance;
if(passwd != "qwerty")
    cout << "wrong password" << endl;
else
    cout << "balance: " << balance << endl;
```

För att förhindra att man skriver utanför avsett minne kan man använda formatväljare för cin:

```
char buf[10];
```

Slide 7

```
// begränsa antalet inlästa tecken
// till storleken på buf
while(cin >> setw(sizeof buf) >> buf)
    do_something(buf);
```

Överlagring av in- och utmatningsoperatorer

- När man vill överföra data från en ström till ett objekt överlagrar man `operator>>`
- När man vill skriva ut ett objekt till en ström överlagrar man `operator<<`

```
// i headerfilen:
```

Slide 8

```
class A
{
    friend ostream &operator<<(ostream &, const A &);
    friend istream &operator>>(istream &, A &);
    int i, j;
};
ostream &operator<<(ostream &, const A &); // deklaration
istream &operator>>(istream &, A &);
```

Slide 9

```

// i implementationsfilen:
ostream &operator<<(ostream &s, const A &a) // definition
{
    return s << "{" << a.i << ", " << a.j << "}";
}
istream &operator>>(istream &s, A &a)
{
    return s >> a.i >> a.j;
}
...

A a;
cin >> a;           // skriv till objektet från stdin
cout << a << endl; // skriv ut objektet till stdout

```

In- och utmatningsiteratörer i STL

- STL-behållare kan man fylla direkt från en ström med en inmatnings-iteratör, `istream_iterator`
- Man kan mata ut innehållet i en behållare till en ström med en utmatnings-iteratör, `ostream_iterator`

Slide 10

```

vector<int> v;
copy(istream_iterator<int>(cin), // hämta från stdin
     istream_iterator<int>(),    // eof-indikator
     back_inserter(v));         // sätt in i slutet

transform(v.begin(),           // beräkna och skriv ut x % 10
          v.end(),
          ostream_iterator<int>(cout, "\n"),
          bind2nd(modulus<int>(), 10)); // x % y, y = 10

```

Filhantering

För att knyta en ström till en fil använder man en filström:

`ifstream` för inmatning eller `ofstream` för utmatning

```
#include <fstream>                // ifstream, ofstream

ifstream input("my_file");        // skapa infil
ofstream output("my_file.sort");  // skapa utfil
vector<int> v;
int i;

if(!input)                        // kontrollera att filerna gick att öppna
    cout << "open failed for input file" << endl;
if(!output)
    cout << "open failed for output file" << endl;
```

Slide 11

```
while(input >> i)                  // hämta data
    v.push_back(i);                // lägg i vektorn
```

```
sort(v.begin(), v.end());          // sortera
```

```
vector<int>::iterator it;          // skapa iterator
for(it = v.begin(); it != v.end(); it++)
    output << *it << endl;         // iteratorn ger elem
```

Slide 12

Man kan bestämma hur en fil ska öppnas (lägg till, skriv, läs):

```
#include <iostream>    // ios_base
#include <fstream>     // ofstream, fstream
```

Slide 13

```
ofstream s1("result", ios_base::app);
fstream s2("temp", ios_base::in | ios_base::out);
```

Olika åtkomsttyper:

app – lägg till (*append*), **ate** – finn slutet (*at end*), **binary** – binär,
in – läs, **out** – skriv, **trunc** – töm fil (*truncate*)